

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California



### Core Validation for Maritime Operations Simulation (MarOpsSim)

by

Arnold Buss  
Thomas Halwachs

November 1999

Approved for public release; distribution is unlimited.

Prepared for: US Coast Guard Research and Development Center  
Maritime Operations Technology Division  
1082 Shennecossett Rd.  
Groton, CT 06340

DTIC QUALITY INSPECTED 4

19991220 019

NAVAL POSTGRADUATE SCHOOL  
MONTEREY, CA 93943-5000

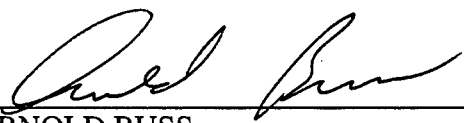
RADM Robert C. Chaplin  
Superintendent

Richard Elster  
Provost

This report was prepared for and funded by the United States Coast Guard Research and Development Center, Maritime Operations Technology Division, 1082 Shennecossett Rd., Groton, CT 06340.

Reproduction of all or part of this report is authorized.


This report was prepared by:

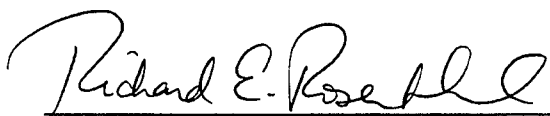
  
ARNOLD BUSS  
Assistant Professor of  
Operations Research

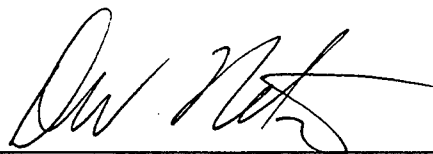
  
THOMAS HALWACHS  
Associate Provost for  
Information Technology

Reviewed by:

Released by:

  
R. KEVIN WOOD  
Associate Chairman for Research  
Department of Operations Research

  
RICHARD E. ROSENTHAL  
Chairman  
Department of Operations Research

  
DAVID W. NETZER  
Associate Provost and Dean of Research

**REPORT DOCUMENTATION PAGE**

Form approved

OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

**1. AGENCY USE ONLY (Leave blank)****2. REPORT DATE**

November 1999

**3. REPORT TYPE AND DATES COVERED**

Technical

**4. TITLE AND SUBTITLE**

Core Validation for Maritime Operations Simulation (MarOpsSim)

**5. FUNDING**

MIR NO. DTCG39-99-XADW217

**6. AUTHOR(S)**

Arnold Buss and Thomas Halwachs

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**Naval Postgraduate School  
Monterey, CA 93943**8. PERFORMING ORGANIZATION  
REPORT NUMBER**

NPS-OR-00-003

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**United States Coast Guard Research & Development Center  
Maritime Operations Technology Division  
1082 Shennecossett Rd.  
Groton, CT 06340**10. SPONSORING/MONITORING  
AGENCY REPORT NUMBER****11. SUPPLEMENTARY NOTES**

The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

Approved for public release; distribution unlimited.

**12b. DISTRIBUTION CODE****13. ABSTRACT (Maximum 200 words)**

The Maritime Operations Simulation (MarOpsSim) is a modern discrete event simulation being used by the US Coast Guard. MarOpsSim was originally developed by the US Coast Guard Research & Development Center to examine elements of the Search & Rescue and Law Enforcement missions. MarOpsSim is now being extended to encompass all fourteen Deepwater missions to support the Deepwater Acquisition. This document reports the results of the Core Verification and Validation (Core V&V) effort conducted at the Naval Postgraduate School by the authors. Five key core elements were tested: Random Number Generation, Event List Processing, Platform Movement, Sensing, and Script Parsing. Following rigorous tests, MarOpsSim is deemed to have had its core functionality validated.

**14. SUBJECT TERMS**Discrete-Event Simulation; United States Coast Guard; Deepwater Acquisition;  
Research & Development Center**15. NUMBER OF  
PAGES**

35

**16. PRICE CODE****17. SECURITY CLASSIFICATION  
OF REPORT**

Unclassified

**18. SECURITY CLASSIFICATION  
OF THIS PAGE**

Unclassified

**19. SECURITY CLASSIFICATION  
OF ABSTRACT**

Unclassified

**20. LIMITATION OF  
ABSTRACT**

UL

# **Core Validation for Maritime Operations Simulation (MarOpsSim)**

**Arnold Buss  
Thomas Halwachs**

## **Executive Summary**

The Maritime Operations Simulation (MarOpsSim) is a modern discrete event simulation being used by the US Coast Guard. MarOpsSim was originally developed by the US Coast Guard Research & Development Center to examine elements of the Search & Rescue and Law Enforcement missions. MarOpsSim is now being extended to encompass all fourteen Deepwater missions to support the Deepwater Acquisition Project. As part of the MarOpsSim development effort, this document reports the Core Verification and Validation (V&V) of MarOpsSim.

Core V&V consists of testing the elements that are fundamental to the execution of the low-level simulation algorithms. Rather than validating the modeling of specific platforms (cutters, aircraft, etc.), it emphasizes validation of basic functionality. Since modeling of specific assets, capabilities, and tactics is done in terms of the core elements, Core V&V is the first step in the overall Verification and Validation process.

The Core V&V consisted of verifying and validating five disparate elements of MarOpsSim: (1) Random Number Generation; (2) Event List Processing; (3) Platform Movement; (4) Sensors; and (5) Script Parsing.

Each element was tested thoroughly using simple notional scenarios. The purpose of each scenario was to isolate the individual element to the greatest degree possible so that the "true" outcomes could be pre-determined and the simulation results compared with the "true" ones. When appropriate, statistical tests were performed to support the conclusions.

Random Number generation is the most fundamental ingredient to any stochastic simulation model. The results of such a model can only be as good as the underlying random number generator. MarOpsSim uses a modified multiplicative congruential generator that was subjected to a full array of tests since it had previously undergone

limited testing. Visual inspection of histograms and scatter plots did not reveal any obvious deviations from uniformity. Statistical tests revealed that the generated values can be treated as independent identically distributed uniform random variates.

MarOpsSim's Event List processing was shown to correctly schedule events on the Future Events List, correctly remove current events to the Current Event List, and correctly process each event as it occurred.

MarOpsSim's platforms all use uniform linear motion as the movement algorithm and use parameters to distinguish different platforms. Air and surface assets are differentiated further by the restricted movement of surface assets to water areas. Tests of restricted and unrestricted platform movements showed that the linear motion algorithm was correctly implemented in MarOpsSim. Platforms reached their designated waypoints and destinations after the correct time delays. Tests of restricted movement demonstrated that MarOpsSim's Land Avoidance algorithm correctly identified waypoints around script defined obstructions (land, shoal water, and/or restricted airspace) and the platforms correctly visited these waypoints en route to their ultimate destinations.

MarOpsSim uses a common algorithm for all sensors based on Range/Probability curves for each sensor. These Cumulative Probability of Detection (CPOD)/Range curves are further modified by such factors as visibility and sea state, which in turn are generated based on the season. Different sensors may be adequately modeled with this approach, since there are many degrees of freedom and the CPOD/Range curves may be configured to closely match any particular sensor characteristics. Tests using both stationary and moving targets showed that MarOpsSim's sensor algorithm is correctly implemented and that its probability of detection are properly modified by weather.

Scripts are parsed into database commands that are executed throughout the program. In all the runs performed, the script parser correctly translated the different script files into commands in the scripts database and that these commands were correctly executed by the MarOpsSim program.

The basic core functions listed above have been thoroughly tested and are operating as designed. The execution of the core functionality of MarOpsSim, a critical first step towards the ultimate V&V of the MarOpsSim model for Deepwater Acquisition applications, is correct and can be considered complete.

# 1 Introduction

This report documents the Core Validation of the Maritime Operations Simulation (MarOpsSim) model. MarOpsSim was originally developed by the US Coast Guard Research & Development Center to model basic elements of the Search & Rescue and Law Enforcement missions. MarOpsSim is now being extended to encompass all fourteen Deepwater missions to support the Deepwater Acquisition Project. Deepwater will use MarOpsSim as a decision support and comparison tool as part of their acquisition evaluation strategy.

Core Validation is concerned with aspects of MarOpsSim that form the foundation for all scenarios and, ultimately, the use of the model. At issue are the basic features that comprise a simulation model. The Core Validation is concerned with whether these fundamental tasks are correctly performed in MarOpsSim so that more complex models may be fashioned.

The Core Validation features to be tested are as follows:

1. Random Number generation
2. Event List Processing
3. Platform Movement
4. Sensors and Weather
5. Script Parsing

Random Number Generation (or pseudo-random number generation) is an algorithm that produces streams of numbers that behave like independent, identically distributed random variates. The statistical properties of a simulation's Random Number generator are critical to the correct performance of all features depending on randomness. In MarOpsSim, those features are mostly concerned with sensors and weather. The Event List processing is the most fundamental aspect of a discrete event simulation model, since all behaviors of entities rely on the events being correctly processed. For models concerned with surface and air assets, the platform movement algorithm is important, as is the sensing model.

MarOpsSim is implemented using four groups of components: (1) The core model is responsible for Event List management, Random Variate generation, and various utility functions. It is implemented in Visual C++ and is an executable file. (2) Scripts are written in a unique MarOpsSim scripting language as plain text files. The scripts are written to establish the model set up including experiment parameters, geography, weather as well as all platform (surface or air, asset or target) characteristics, capabilities, allocation and tactical behavior. The script parser is responsible for transforming the script files into commands in the input database. (3) The script parser is likewise implemented in Visual C++ and is an executable file. (4) The Tactical Functions Library contains a number of commonly used functions.

## 2 Verification of Random Number Generator

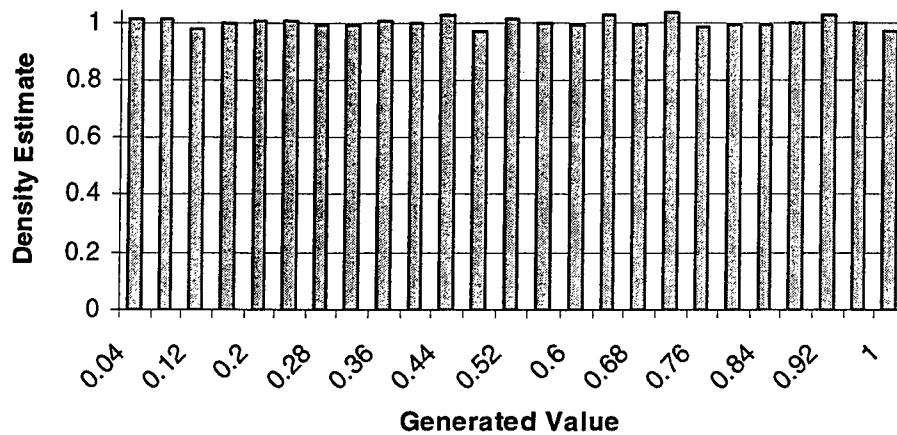
This section will discuss the algorithm used by MarOpsSim for generating random variates followed by the tests conducted. Random variate generators must behave as if the generated values were uniformly distributed between 0 and 1 and as if they were statistically independent. Statistical tests for uniformity and for independence showed that the MarOpsSim random number generator provides well-behaved streams—that is, sequences of numbers that are statistically indistinguishable from independent, identically distributed Uniform (0, 1) random variates.

### 2.1 Algorithm

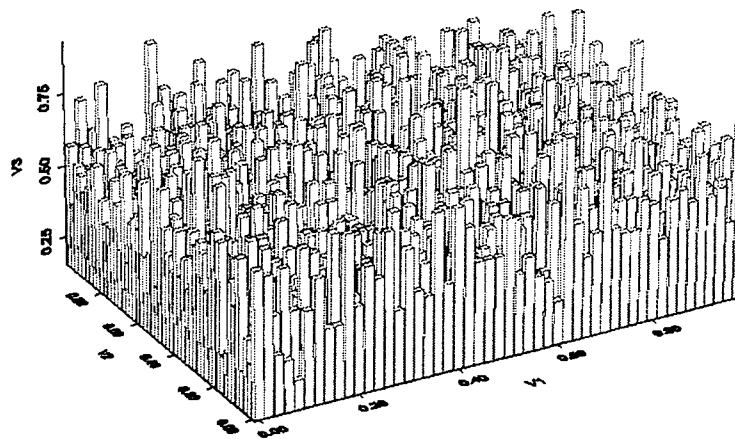
MarOpsSim uses a modified form of a multiplicative congruential generator, the most common form of pseudo-random generators. Although widely used, such generators must be used with care, since poorly chosen parameters can produce pseudo-random numbers with undesirable properties (Law and Kelton, 1992; Knuth, 1998). Based on the statistical tests described below, the random number generator used by MarOpsSim uses well-chosen parameters that produce sequences of pseudo-random variates that fit the criteria for uniformity and independence quite well.

The two measures of whether a pseudo-random generator is adequate involve whether the numbers are indeed uniformly distributed between 0 and 1 and whether they appear to be independent. An informal way to examine the uniformity of a stream of numbers is to plot an estimate of its probability density function (pdf), as shown in Figure 1. This plot was obtained by generating 100,000 numbers from the MarOpsSim random number generator and obtaining a frequency histogram with 25 bins equally distributed on [0, 1]. These 100,000 values were generated from the 'Stream 1,' one of four sequences of generated values tested in Table 1 and Table 2 in the following section. The counts were scaled by dividing by the number of observations and multiplying by the bin width (0.04) to produce an estimate of the pdf, which in the case of the uniform distribution is the constant function with value 1. Visual examination of Figure 1 indicates that there do not appear to be any substantial deviations from uniformity.

It is important to establish the uniformity of pseudo-random number generators in higher dimensions. Figure 2 shows a plot of the estimated two-dimensional density from the MarOpsSim random number generator. This plot was obtained from 100,000 random variates grouped into 50,000 pairs. These were the same 'Stream 1' used for the one-dimensional histogram shown in Figure 1. A two-dimensional frequency histogram was obtained with 50 cells in each dimension. The frequency counts were scaled by dividing by the number of observations (50,000 pairs) and multiplying by the cell areas. Again, there is no indication of serious deviation from uniformity.



**Figure 1. Density Estimate from 100,000 Generated Variates**

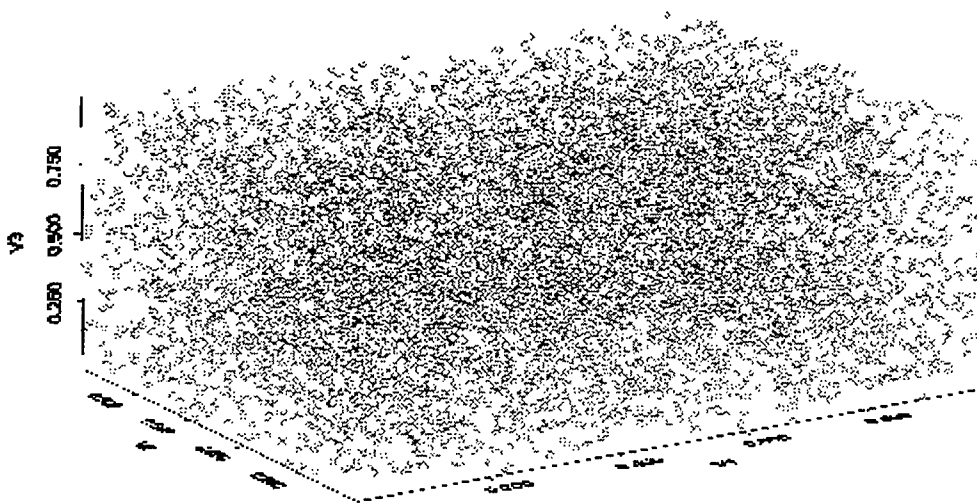


**Figure 2. Two Dimensional Density Plot for Generated Random Variates**

As a final informal test of uniformity, a scatter plot was made for successively generated random numbers taken in triples. Each successive random triple was plotted as if the three numbers were the x-y-z coordinates of a point.

The resulting three-dimensional scatter plot is shown in Figure 3. As can be seen, the points appear to be uniformly scattered throughout the unit cube.





**Figure 3. Scatter Plot in Three Dimensions**

## **2.2 Statistical Tests**

While plots such as Figures 1-3 are useful for informally checking the random number generator, they are no substitute for rigorous statistical tests. Therefore, four tests were conducted on output from the MarOpsSim random number generator: The Kolmogorov-Smirnov (K-S) goodness of fit test in one dimension, and Chi-square goodness of fit tests in one, two, and three dimensions. The K-S and Chi-square test in one dimension are formal validation of the informal observation of the uniformity apparent in Figure 1; the remaining Chi-square tests are, likewise, formal verification of the informal observations of uniformity from Figures 2 and 3.

Four sequences of 100,000 pseudo random numbers were generated in MarOpsSim using scripts provided by the Coast Guard. Each of the sequences used a different starting seed to generate the observations. Two sets of statistical tests were performed on the sequences to evaluate the uniformity (i.e. whether the pseudo random numbers could be reasonably said to be drawn from the uniform probability distribution) and the independence of the sequences.

## **2.3 Uniformity of Generated Variates**

A histogram of 100,000 random variates is shown in Figure 1 and a 2-dimensional histogram of the same data is shown in Figure 2. No substantial deviation from uniformity can be observed. This is confirmed by the statistical tests performed on four sequences of variates drawn from the random number generator.

Four tests were applied to evaluate the uniformity of the sequences: The Kolmogorov-Smirnov (K-S) test in one dimension and  $\chi^2$  goodness of fit tests in one-, two- and three dimensions. The K-S test is more powerful than the  $\chi^2$  test in one dimension, but it has no higher dimension counterpart. It is important to test random number generators in higher dimensions since correlations can sometimes appear.<sup>1</sup>

The number of bins for the  $\chi^2$  tests were: 1-D = 100 bins with 99 degrees of freedom; 2-D = 50 bins per dimension with 2499 degrees of freedom; and 3D = 15 bins per dimension for 3374 degrees of freedom.

The results of the four tests are shown in Table 1. The K-S tests in one dimension and the  $\chi^2$  test in 2 dimensions all give very high p-values. The chi-square tests in 1 and 3 dimensions for the third stream have somewhat lower p values than the others, despite the fact that its K-S statistic is quite good. This suggests that there may be sections of the random number stream with slightly less uniformity than others.

Stream	K-S Test	1-D $\chi^2$	2-D $\chi^2$	3-D $\chi^2$
1	0.0020 (0.82)	98.73 (0.49)	50.65 (1.00)	3384.97 (0.81)
2	0.0024 (0.62)	107.60 (0.26)	48.53 (1.00)	3299.92 (0.82)
3	0.0021 (0.76)	115.36 (0.12)	48.91 (1.00)	3454.83 (0.16)
4	0.0024 (0.62)	80.41 (0.91)	49.17 (1.00)	3356.42 (0.42)

**Table 1. Goodness of Fit Tests for Uniformity of Generated Variates**  
(Numbers in parentheses are p values)

All four sequences pass the uniformity tests. It can be concluded that the MarOpsSim random number generator is producing reasonably uniform values.

## 2.4 Independence of Generated Variates

Two tests for independence were conducted on the MarOpsSim random number generator: the runs test and the autocorrelation test.

The runs test is a nonparametric test of independence based on the distribution of the number of increasing runs (Knuth, 1998). Here, a run is a sequence of values that are

---

<sup>1</sup> The most famous example of this is IBM's RANDU generator that was shipped with all its FORTRAN compilers in the 1960's and 1970's. RANDU was shown to be fatally flawed due to its high correlation in three dimensions, despite the fact that it passed one- and two-dimensional tests. (Law and Kelton, 1992)

increasing (a similar test could be conducted for the runs down). Under mild conditions, the frequency distribution of the number of runs can be tested by a  $\chi^2$  test, as described in Knuth (1998).

A random number generator passes the autocorrelation test if the null hypothesis that autocorrelations up to a given order are not significantly different from zero. Note that zero autocorrelation is, of course, not the same as independence. Under mild conditions, the autocorrelation function is asymptotically standard normal under the null hypothesis of independence. The autocorrelation test was performed for lags 1-4, since it is possible for there to be significant autocorrelation for lags greater than 1.

		Autocorrelation Tests			
Stream	Runs Test	Lag 1	Lag 2	Lag 3	Lag 4
1	7.783 (0.25)	0.0018 (0.87)	-0.0015 (0.90)	0.0073 (0.52)	-0.0021 (0.86)
2	4.641 (0.59)	0.0034 (0.76)	-0.0020 (0.86)	0.0012 (0.92)	0.0033 (0.77)
3	3.811 (0.70)	0.0018 (0.88)	0.0026 (0.82)	0.0012 (0.92)	-0.0015 (0.90)
4	6.720 (0.35)	0.0000 (1.00)	-0.0023 (0.84)	0.0076 (0.50)	0.0017 (0.88)

**Table 2. Tests of Independence for Random Number Generator**  
(Numbers in parentheses are p values)

The results of the tests for independence are shown in Table 2. The Runs Test statistics are shown together with the associated p-values. Estimated autocorrelations are shown with their associated p-values. The MarOpsSim random number generator passed all tests, indicating that there is no concern about significant correlation between successive generated values.

### **3 Verification of Event List Processing**

When building a discrete event simulation model with a special-purpose package for simulation, the event list processing is built-in and may typically be assumed to correctly process events. Since MarOpsSim is implemented in native C++ code, it must implement the event list processing within the core model itself. Therefore, the discrete event processing algorithm must be verified.

MarOpsSim uses a modified discrete event algorithm for advancing time. All scheduled events are stored in a future events list. At each step, simulated time is advanced to that of the next occurring events. All events scheduled to occur at that time are removed and placed in a current event list ("Now" events). Each current event is executed by invoking the script function associated with that event.

MarOpsSim Event List processing was verified by ensuring that the basic Discrete Event algorithm was correctly implemented, that the Future Event List correctly represented the status of all pending events, and that the Current Events List was correctly determined from the Future Event List. In over 100 scenarios, all three (discrete event logic, current events list and future events list) were correctly executed, validating MarOpsSim's Event List processing.

#### **3.1 Discrete Event Logic**

Discrete Event Simulation (DES) methodology advances time non-uniformly, unlike a Time Step Simulation which advances simulated time in fixed increments. DES tends to provide more accurate depictions of complex interactions between entities because events occur when they are scheduled and not coalesced into time buckets. Such artificially simultaneous events in time step simulation models result in arbitrary adjudication logic to determine the ordering of events. These arbitrary tie-breaking rules can have a profound impact on the simulation results.

MarOpsSim contains a feature that enables discrete time increments to be specified to establish the granularity of the clock when the order of simultaneous event is not an issue, with all events in a given time interval occurring at the same simulated time. However, all of the verification runs were performed with that increment set to zero. That ensured the fewest number of simultaneous events. None of the detailed runs examined exhibited any anomalous behaviors or artificially simultaneous events. All simultaneous events were processed as expected.

#### **3.2 Future Events List**

The Future Events list stores all the pending events that have been scheduled at a given time. The maximum number of pending events is specified by the user; there can

never be any more than that number of events on the Future Events list. This should not be a concern for normal scenarios if set sufficiently high.

The event list logic was tested by examining detailed dumps of the future and current event lists for more than 100 scenarios involving a variety of different assets. In all cases the future event list logic correctly removed the correct list of events and updated the simulation clock appropriately. The current event list correctly reflected the removed events in every case. When executed, the event scripts made the correct state transitions and placed the correct additional events on the Future Events list. An example of the state of the future events list, the current events removed and executed, and the resulting updated future events list is shown in Figure 4.

```
Event List: 27.5506
-----
THISTASK: Primary platform: ASSET_EYE, 0, NO Secondary platform , Source: TACTIC, Time: 30.
GO_AROUND: Primary platform: PATROLCUTTER, 2, NO Secondary platform , Source: TACTIC, Time: 30.3417.
-----

Now List:
-----
THISTASK: Primary platform: ASSET_EYE, 0, NO Secondary platform.
-----

Event List: 30
-----
THISTASK: Primary platform: ASSET_EYE, 0, NO Secondary platform , Source: TACTIC, Time: 40.
GO_AROUND: Primary platform: PATROLCUTTER, 2, NO Secondary platform , Source: TACTIC, Time: 30.3417.
-----

Now List:
-----
GO_AROUND: Primary platform: PATROLCUTTER, 2, NO Secondary platform.
-----

Event List: 30.3417
-----
THISTASK: Primary platform: ASSET_EYE, 0, NO Secondary platform , Source: TACTIC, Time: 40.
GO_AROUND: Primary platform: PATROLCUTTER, 2, NO Secondary platform , Source: TACTIC, Time: 30.9686.
-----
```

**Figure 4. Portion of Current ('Now') and Future Event List**

Figure 4 shows snapshots of the event list for a scenario that tests the land avoidance algorithm that covers time 27.5506 through 30.3417. After each event is processed, the algorithm must determine the next event that occurs in time. At each step the "Now List" consists of all events occurring at a given time. As shown in Figure 4, there are two events, at time 27.5506, on the list. MarOpsSim correctly chooses THISTASK as the next event to process, as reflected in the "Now List." Simulated time is then correctly advanced to time 30, the scheduled time of the event, and the subsequent scheduled event (another THISTASK by the ASSET\_EYE) scheduled for time 40, as indicated by the updated event list. At time 30, MarOpsSim again correctly chooses the nearest event, this time the second on the list, a GO\_AROUND event by the PATROLCUTTER, even though it is actually listed second on the output. Simulated time is again correctly updated to the time of that event's occurrence, this time to 30.3417.

The event list processing algorithm is thus verified to be correctly processing events in this scenario. Furthermore, in over 100 scenarios run in the course of testing MarOpsSim features, such as platform movement and sensors, the Event List processing was always correctly performed. The Future Events List has therefore been verified to correctly schedule future events and correctly execute them in temporal order.

### **3.3 Current Events List**

At a given simulated time, the Current Events List consists of all events removed from the future events list scheduled to occur at that time. The user specifies the maximum number of simultaneous events that can occur, thereby limiting the extent to which ties are allowed. If the scenario exceeded that value, then the simulation run terminates with an error. The list size can be specified in the scenario configuration to allow an application to be appropriately scaled. This can be set sufficiently high and should not be a concern for normal scenarios.

In simple scenarios, such as that of the previous section, the "Now List" was always correctly identified by the MarOpsSim event list algorithm. This is important because of the manner in which simulated time proceeds in discrete event simulation, by the earliest scheduled time of all events. As seen in the event list outputs, the events are not always listed in temporal order. Therefore, the algorithm that determines the earliest set of events ("Now List") is crucial.

For example, in Figure 4 the Current Event List ("Now" List) is correctly identified at each event step. In over 100 scenarios examined, the Current Events List was always correctly extracted from the Future Events List. That is, all events that were scheduled to occur with simultaneous simulated time were correctly identified, removed from the Future Events List, and processed. The Current Event Lists algorithm is deemed to be correct.

## 4 Verification of Platform Movement

Platforms move according to uniform motion that is “linear,” subject to a curved earth model described below. Such movement is implemented in MarOpsSim by the “GoToWaypoint” tactic for surface craft and “FlyToWaypoint” for aircraft. The following platforms were modeled in MarOpsSim scenarios tested:

- GoFast
- SailBoat
- MedPowerBoat
- IslandA
- HC-130-1500

All platforms use the same movement algorithm, which is based on uniform linear motion. The only differences between the movement of the different assets are the speeds at which they travel, each platform specifying low, medium, and maximum speeds. Thus, there is a single unrestricted movement algorithm that serves all platforms, and it is this algorithm that was evaluated in the core V&V phase. There is also a single restricted movement algorithm which can be used by surface platforms for land avoidance, shallow water avoidance, etc., and used by aircraft for restricted airspace avoidance. Land avoidance scenarios were used to evaluate this algorithm.

All unrestricted platform trajectories correctly moved to their respective destinations. The correct amount of time elapsed between the onset of a move and the arrival at a waypoint or destination, and the correct destination or waypoint was in fact achieved. All the restricted trajectories correctly negotiated around the land mass barrier that blocked its path by correctly identifying the boundary waypoints and traveling to them in succession. Therefore, the movement part of MarOpsSim is correctly implemented and is considered validated for the uniform linear motion implemented.

### 4.1 Unrestricted Movement Algorithms

In order to evaluate the movement logic alone, each platform was first examined without any interaction with barriers, such as land for surface assets. The times of occurrence of the key platform events, arrival at way points, getting underway, etc, were compared with the theoretical times computed by hand using the curved earth model used by MarOpsSim.

The unrestricted movement algorithms for the four surface assets listed above were tested using several scenarios. In each scenario an asset was moved in a sequence of way points. At each point the location and bearing reported by the simulation was compared to the correct values, computed by hand. Additionally, the transit times for each leg as reported by the scenario were compared with the correct values, again computed by hand.

In all cases the unrestricted movement events were correctly scheduled. This implies that MarOpsSim's movement algorithm (with the latitude correction) is correctly implemented. All surface assets had correct transit times, locations, and bearings in all scenarios tested.

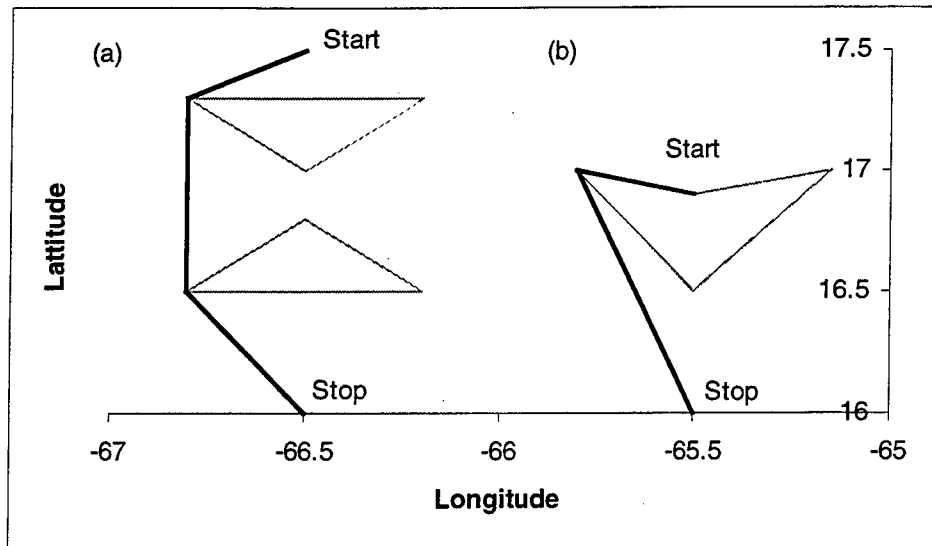
## 4.2 Restricted Movement Algorithms for Surface Assets

Surface assets are normally restricted to move on the water. MarOpsSim models this restricted movement using a land avoidance algorithm. Each land mass is modeled as a sequence of line segments that a surface asset is unable to cross. When a surface asset is commanded to move to a point (or a waypoint) that has a land mass blocking it, the land avoidance algorithm determines another sequence of waypoints to achieve the objective with the minimum distance. For example, if the land mass is an island, then the possible additional waypoints to avoid it are the vertices of the closed polygon consisting of the island's segments. If the surface asset's destination is in the interior of a land mass, the operator is notified and processing continues.

Four sets of runs were performed to test the land avoidance algorithm: (1) Simple triangular islands where the asset had to negotiate two islands between it and its destination; (2) An island with a concave region where the surface asset had to maneuver out of the concave "bay"; (3) Puerto Rico where the asset had to move to a point on the opposite side of the island; and (4) A "figure eight" with Mona Pass and Puerto Rico where the surface asset had to negotiate a series of waypoints around the island through the Mona Pass. In each case the asset was simply commanded to move to a waypoint that was blocked by a land mass; it was the responsibility of the land avoidance algorithm to keep the asset in the water.

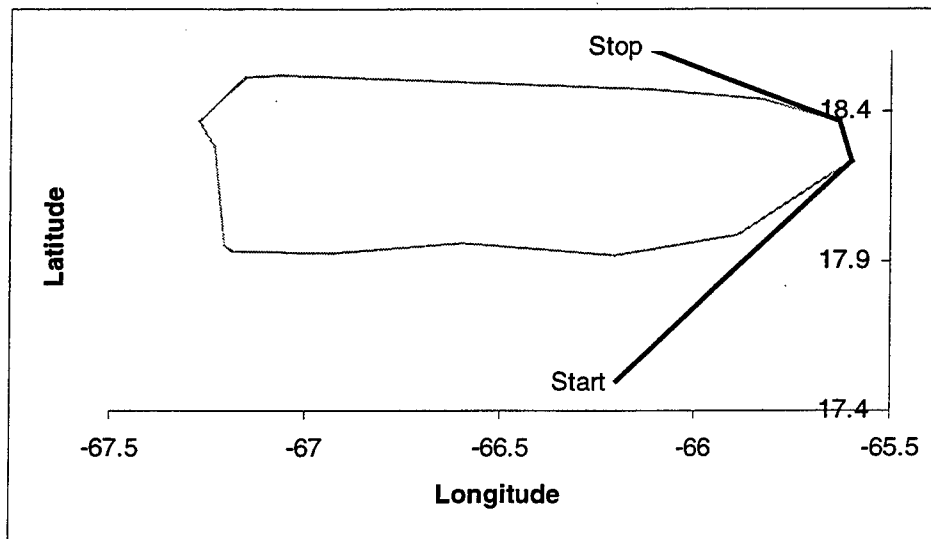
Representative results of the first two runs are shown in Figure 5. The simple triangular islands, shown on the left, are avoided by the trajectory shown from (-66.5, 17.5) to (-66.5, 16.0) [17.5°N 66.5°W to 16.0°N 66.5°W]. Other runs using various starting and ending points produced similar results. In every case, the correct land mass vertices were identified and added to the asset's way point list. The right of Figure 5 shows the concave region with the asset finding its way out of the "bay" in the island (the concave region). The asset starts at 16.9°N 65.5°W and is given 16°N 65.5°W as its destination. The land avoidance algorithm correctly finds the route that achieves the destination while staying in the water. In addition to demonstrating the correctness of the land avoidance algorithm, this example illustrates that convexity of the land masses is not required for it to work properly.





**Figure 5. Land Avoidance of Triangular Islands; and Simple Concave Region**

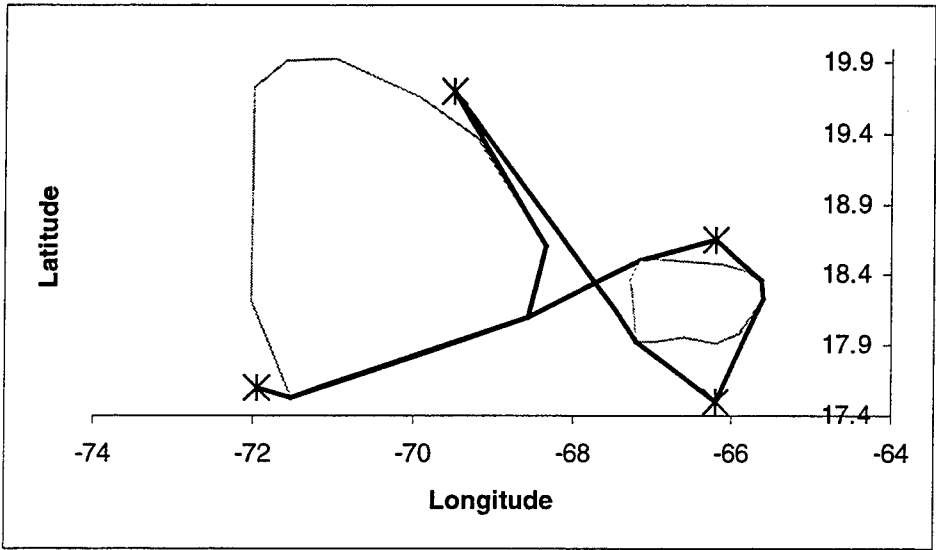
Actual land masses are more complicated than the simple geometric figures in Figure 5. To test the avoidance of such a land mass with more segments, a surface asset was started south of Puerto Rico and given a destination north of Puerto Rico. Figure 6 shows that the land avoidance algorithm correctly negotiates the asset around the land mass, correctly arriving at the two way points on the eastern part of the island.



**Figure 6. Avoidance of Puerto Rico by Surface Asset**

Finally, to illustrate more complex and realistic restricted land movement, Figure 7 shows the trajectory of a surface asset through the Mona Pass and around Puerto Rico. Only the four waypoints indicated were given to the asset, and the land avoidance

algorithm was able to determine correct waypoints that enabled the asset to reach its destination and avoid both land masses.



**Figure 7. Complex Avoidance Around Puerto Rico through the Mona Pass**

## 5 Verification of Sensors and Weather

Since the behavior of sensors in MarOpsSim is affected by weather state, they were considered together. As with movement, there is a single, generic model for all sensors. Changing parameters causes one or another particular sensor to be modeled.

The sensing algorithm works from a Cumulative Probability of Detection (CPOD)/Range Curve, which can be affected not only by the weather state, but by the size and type of the target. Regardless of all these factors, however, the basic algorithm is identical. Furthermore, the algorithm is sufficiently general and flexible that it can be used to capture, or at least approximate any Cumulative Probability of Detection/Range characteristic. The core validation is therefore only concerned with the implementation of the algorithm and not with the modeling of particular sensors.

The Cumulative Probability of Detection/Range curve is approximated by a piecewise linear cumulative probability distribution based on the aforementioned factors. For each sensor-target engagement a draw from that cumulative distribution function (CDF) is made, resulting in the effective range of the sensor for that engagement. If the target enters the randomly drawn range it is considered detected by the sensor. If it does not enter that range during the engagement it is not detected.

Runs were performed with a single “generic” sensor against a target without weather effects and with weather effects. Using 1,000 replications per scenario, the distance of the target from the sensor at detection was recorded. For each scenario three orientations and several target aspect angles were used. Runs were made with the sensor stationary and the target moving and with the target stationary and the sensor moving. The resulting empirical CDF was found to fit well with the theoretical CPOD/Range curve in each instance. Since weather affects the CPOD/Range curve by degradation in exactly the same qualitative manner, it was only necessary to compare the empirical CPOD/Range curve with the theoretical one under a few conditions. In all cases the weather correctly degraded the sensor’s detection.

### 5.1 Sensors with No Weather

A generic sensor was modeled having two range bands in its Cumulative Probability of Detection (CPOD)/range curve. Table 3 and Figure 8 show this sensor information. Two sets of Cumulative Probability of Detection values are included, CPOD and CPOD’. CPOD represents the probability that the initial detection occurs at the given range or closer, and CPOD’ represents the probability that the initial detection occurs at the given range or greater.

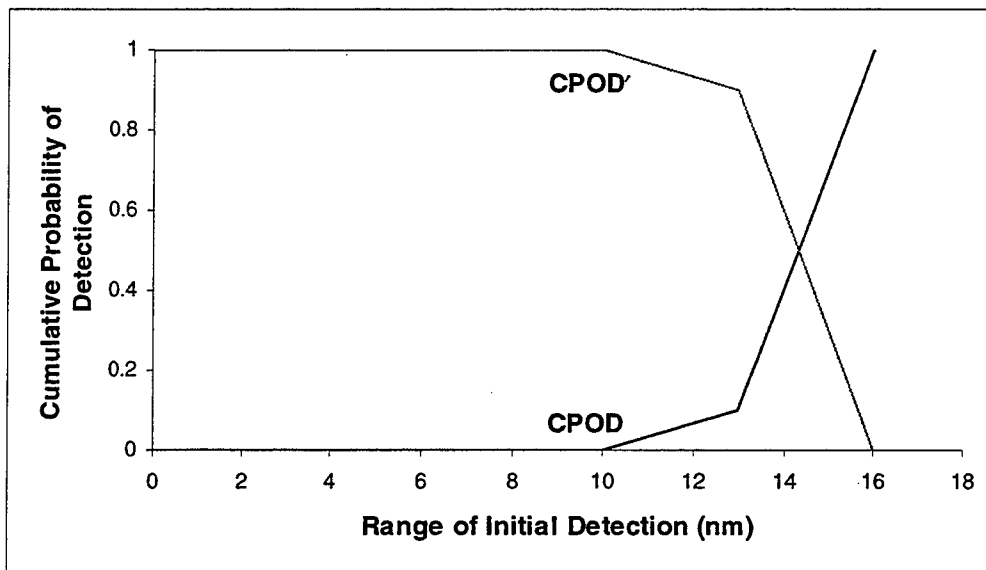
Note that:  $CPOD = 1 - CPOD'$

The data actually entered into MarOpsSim is in the form of CPOD’. Equivalent analysis can be conducted in either the CPOD or CPOD’ domain. Figures 10, 12, 14, 16,

18, 20 and 22 contain CPOD plots, while Table 5 and Figure 23 are in the form of CPOD'.

CPOD	CPOD'	Range of Initial Detection (nm)
0.0	1.0	10
0.1	0.9	13
1.0	0.0	16

**Table 3. Cumulative Probability of Detection vs Range for Tested Sensor**



**Figure 8. Cumulative Probability of Detection vs Range for Tested Sensor**

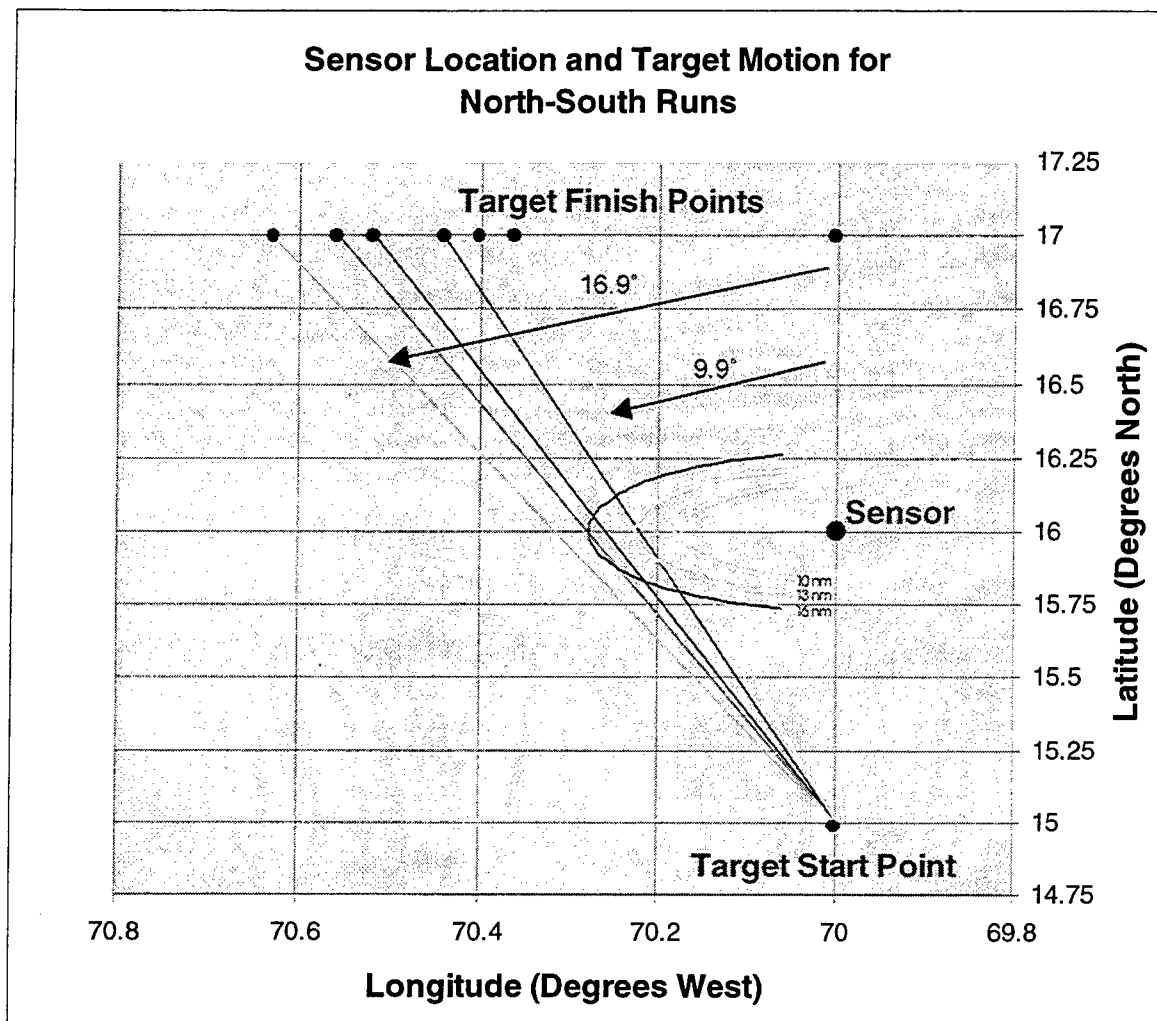
Two sets of runs were conducted for evaluating the MarOpsSim implementation of sensors without Weather: The first set involved a stationary sensor and a moving target; the second a stationary target and a moving sensor. Each set was run with three different orientations of the relative trajectories: North-South, East-West, and NNW-SSE (oblique). For each of these up to seven different trajectories were considered, ranging from the target heading directly toward the sensor (or vice versa) to an angle that just missed the sensor's maximum possible range. Each set of parameters was replicated 1,000 times, with the number of detections recorded along with their range when detected. For the target heading directly toward the sensor, for example, one would expect to observe approximately the same Cumulative Probability of Detection/Range curve as in Figure 8. As the angle increases, so should the range at which the target is detected. The probability of detection should decrease as well.

## 5.2 Stationary Sensor, Moving Target

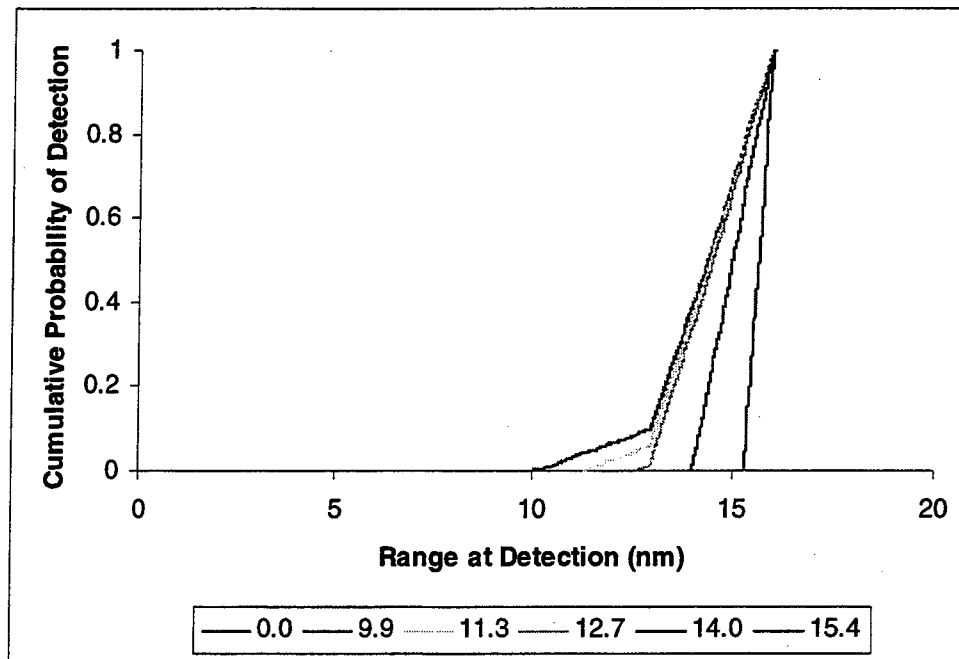
Initial experiments indicated that only the relative position of the target and sensor mattered. Consequently, for the longer runs with a stationary sensor and moving target the sensor was positioned at  $16^{\circ}\text{N } 70^{\circ}\text{W}$  and the target started at  $15^{\circ}\text{N } 70^{\circ}\text{W}$ ,  $15.134^{\circ}\text{N } 69.5^{\circ}\text{W}$ , and  $16^{\circ}\text{N } 71^{\circ}\text{W}$  for the North-South, oblique, and East-West runs, respectively.

### 5.2.1 North-South Runs

As mentioned above, 1,000 replications were made for each of seven target trajectories moving past the sensor. Figure 9 provides an illustration of the North-South target trajectories relative to the sensor and range radii of 10, 13, and 16 nautical miles (nm) around the sensor. The seven trajectory offset angles that were used include 0, 9.9, 11, 12, 14, 15 and 16.9 degrees. The trajectory offset angle is the angle between the target's trajectory and the line segment between the target and the sensor. For example, '0.0' means that the target is heading directly toward the sensor. In this case, the target started at  $15^{\circ}\text{N } 70^{\circ}\text{W}$  and ended at  $17^{\circ}\text{N } 70^{\circ}\text{W}$ . The curve for '9.9' means that the target's trajectory was  $9.9^{\circ}$  offset from the sensor, etc. The 9.9 and 16.9 degree offset angles are labeled in Figure 9. Figure 10 shows the estimated Cumulative Probability of Detection/Range curves, as a function of six target trajectories, for all observed detections.

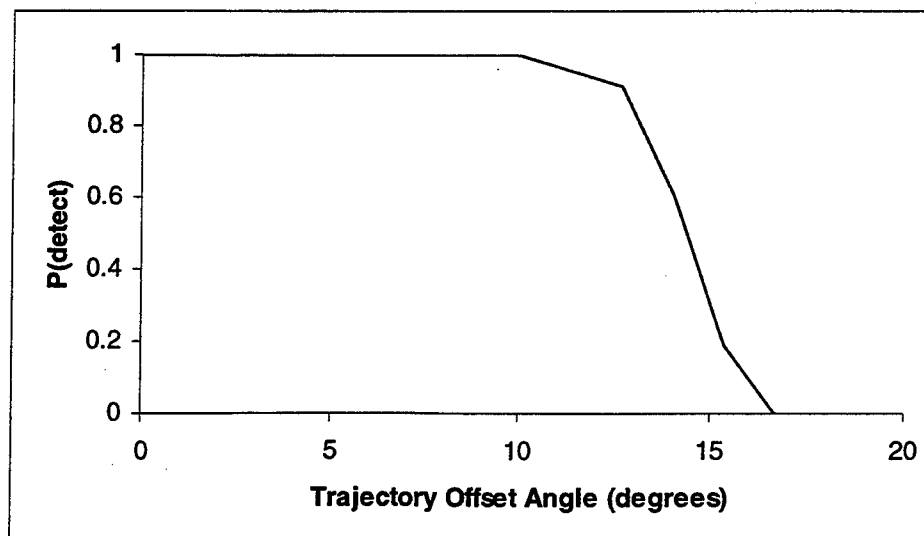


**Figure 9. Target Trajectories for Sensor/Target Runs**



**Figure 10. Estimated Cumulative Probability of Detection vs Range for North-South Scenario**

In Figure 10 the curves for  $0.0^\circ$  and  $9.9^\circ$  overlap nearly completely, since the Cumulative Probability of Detection/Range curves are nearly identical for the two. As the offset increased, the minimum range at which the target was detected increased to the point where the innermost range band was never hit. The final trajectory, an offset of  $16.9^\circ$ , is not shown since there were no detections for that scenario.



**Figure 11. Probability of Detection vs Offset Angle for North-South Runs**

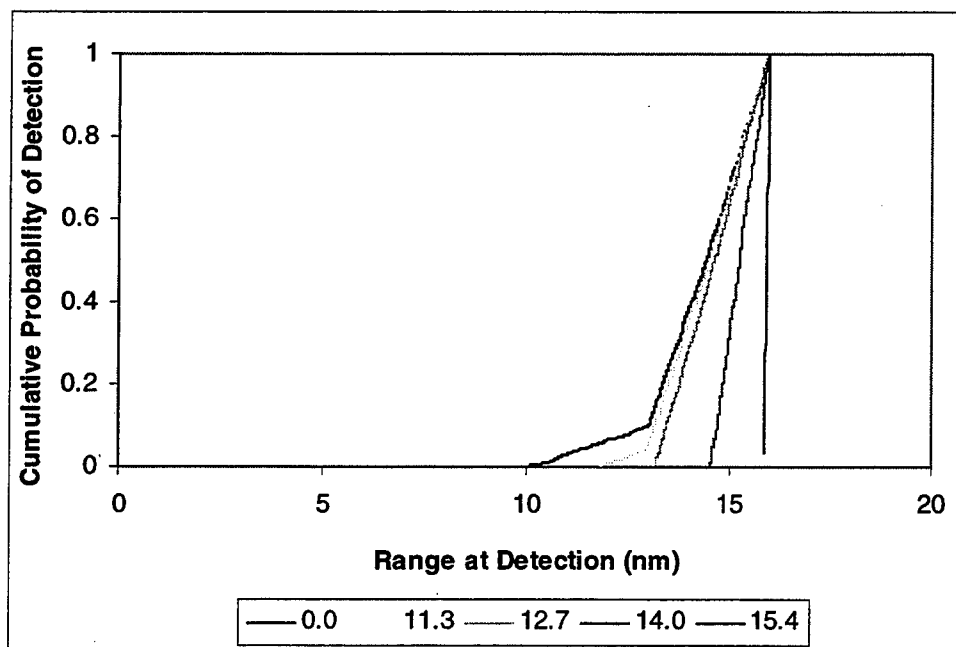
The same runs were used to estimate the probability of detection as a function of trajectory offset angle. This curve is shown in Figure 11.

Both Figure 10 and Figure 11 show excellent agreement with the theoretical probabilities. For example, in Figure 10 the curve for  $0.0^\circ$  is extremely close to Figure 8, the theoretical Cumulative Probability of Detection/Range curve. The other curves are likewise very close matches with the exact ones. Similarly, Figure 11 shows close agreement with the theoretical curve as a function of the trajectory offset angle.

### 5.2.2 East-West Runs

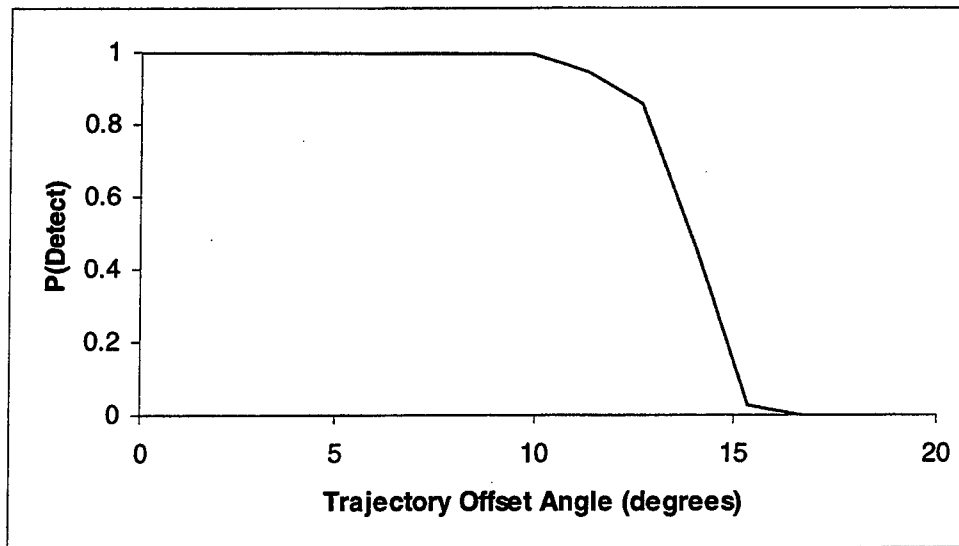
In order to ensure that the North-South results were not an anomaly from the North-South trajectory, an East-West scenario was likewise run. In the base case the target started at  $16^\circ\text{N } 71^\circ\text{W}$  and moved to  $16^\circ\text{N } 69^\circ\text{W}$ , the sensor still being located at  $16^\circ\text{N } 70^\circ\text{W}$ . Each scenario was replicated 1,000 times.

The results of these runs are shown in Figure 12 and Figure 13. As with the North-South runs, there is a close match between the observed and the expected curves.



**Figure 12. Estimated Cumulative Probability of Detection vs Range for East-West Scenario**



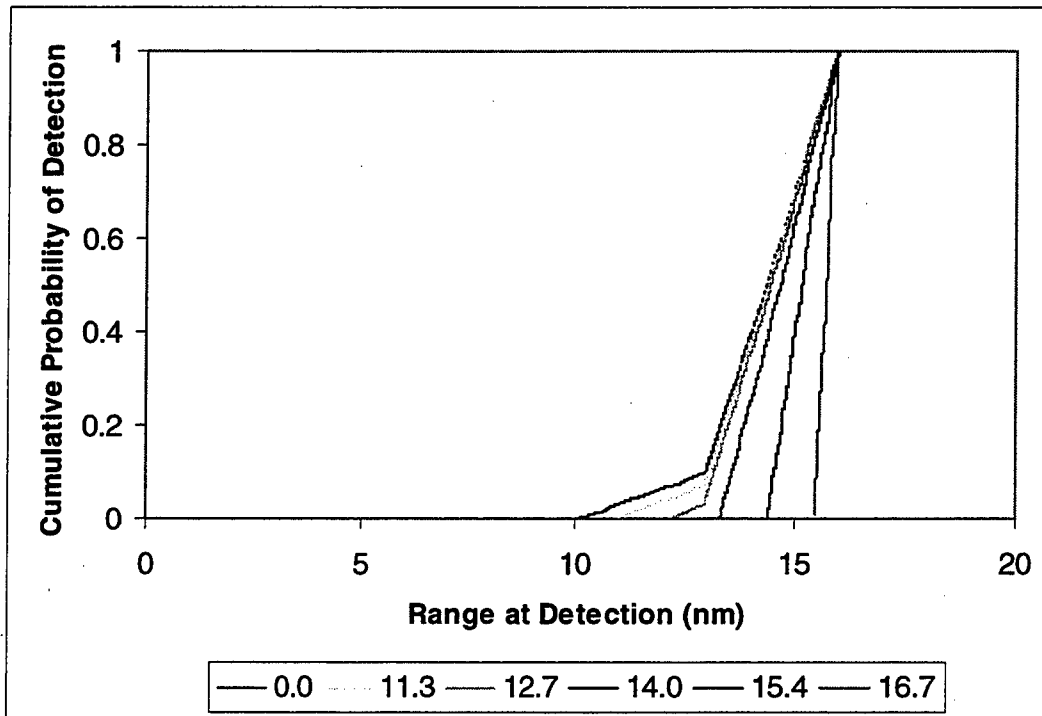


**Figure 13. Probability of Detection vs Trajectory Offset Angle for East-West Scenario**

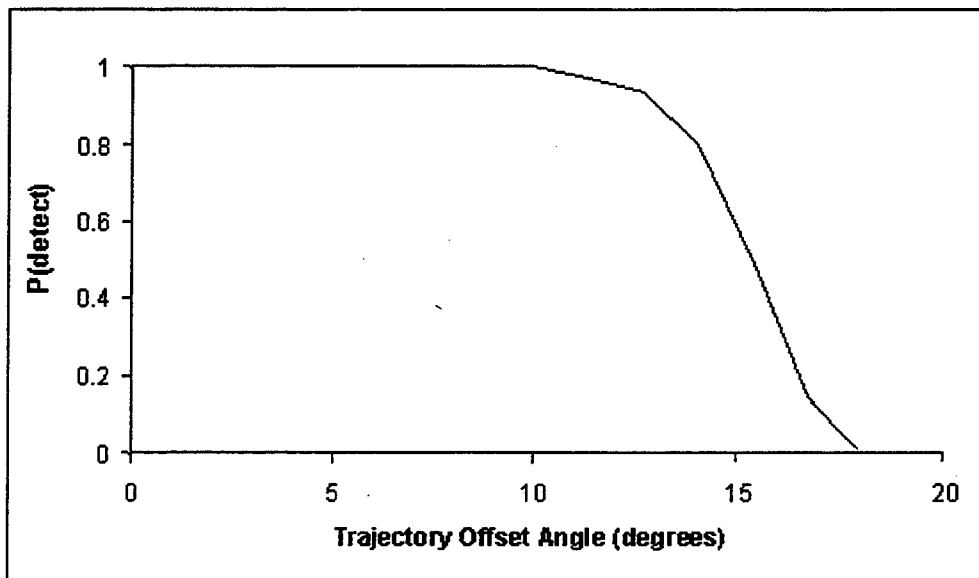
### 5.2.3 Oblique Runs

To ensure that the previous results were not a consequence of the target traveling “parallel” to the axes (bearing North or East, respectively), a final set of runs were performed with the base case being the target starting at 15.134°N 69.5°W and moving to 16.866°N 70.5°W. Comparable runs were performed with the target having an increasingly large angle off the sensor. As in the previous examples, 1,000 replications were performed for each scenario.

The results are shown in Figure 14 and Figure 15 and demonstrate that MarOpsSim’s sensing algorithm is working properly at oblique angles and not just for North-South or East-West trajectories.



**Figure 14. Estimated Cumulative Probability of Detection vs Range for Oblique Scenario**



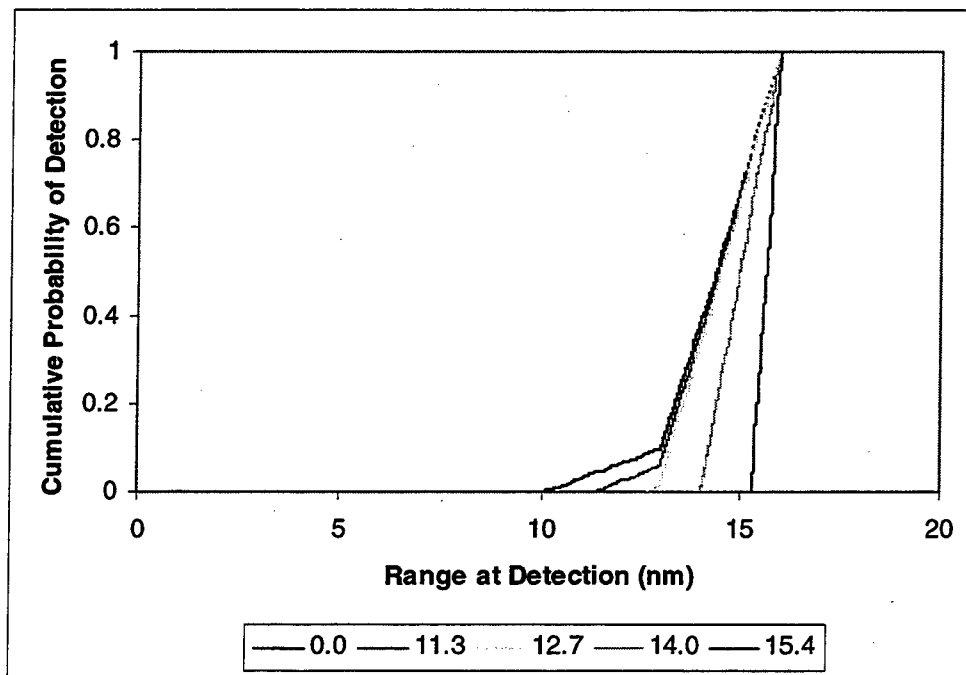
**Figure 15. Probability of Detection vs Trajectory Offset Angle for Oblique Scenario**

### 5.3 Stationary Target, Moving Sensor

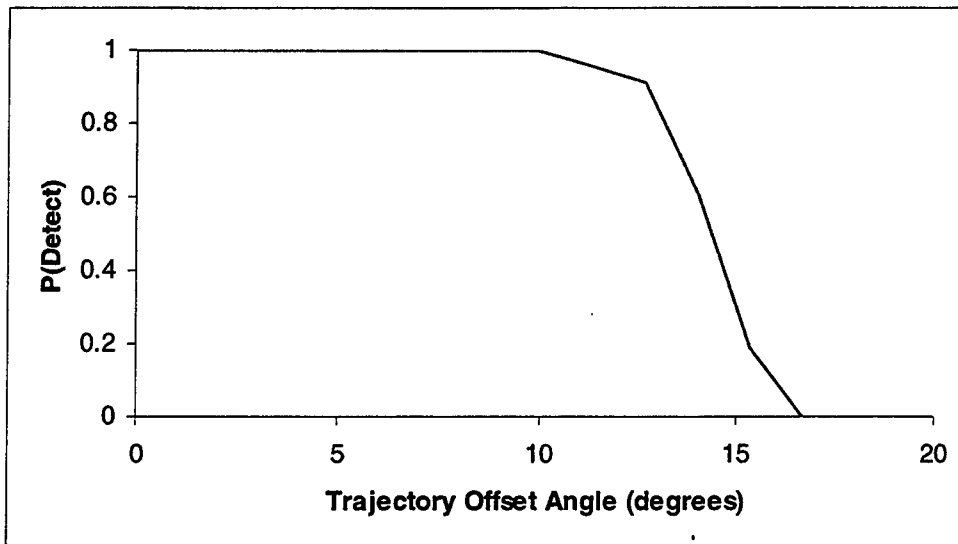
Since a target may just as likely be stationary and be detected by a moving sensor as the other way around, the entire set of runs for the stationary sensor and moving target were run with the roles reversed. For example, in the North-South scenario, the target was stationary at 16°N 70°W and the sensor started at 17°N 70°W and moved to 15°N 70°W. Since the relative motion of the sensor and target were identical, the results should match those of the previous section. As with those runs, each scenario was replicated 1,000 times.

#### 5.3.1 North-South Runs

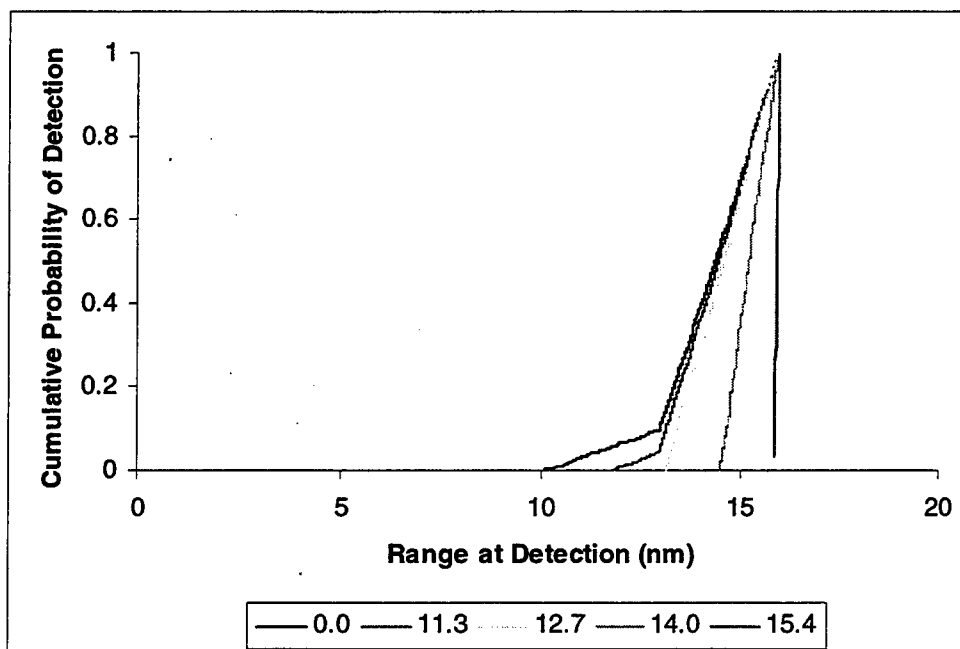
Each of the three scenarios of Section 5.2 was repeated with the roles of sensor and target reversed. The estimated Cumulative POD/Range curves and POD/Offset Angle curves are shown in Figure 16 through Figure 21. These show close agreement with the stationary sensor/moving target, demonstrating that the sensing algorithm works correctly whether the sensor or the target is moving.



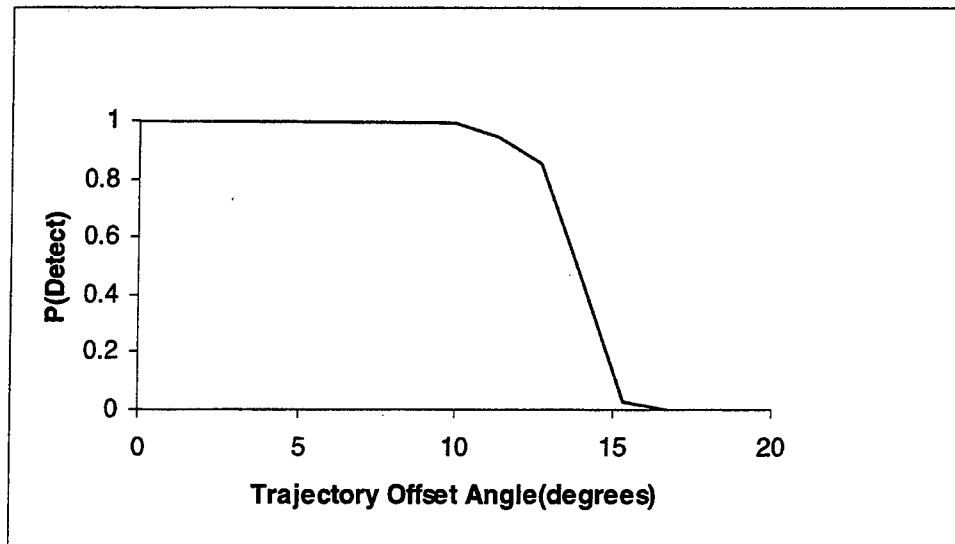
**Figure 16. Estimated Cumulative Probability of Detection vs Range for Moving Sensor, North-South Scenario**



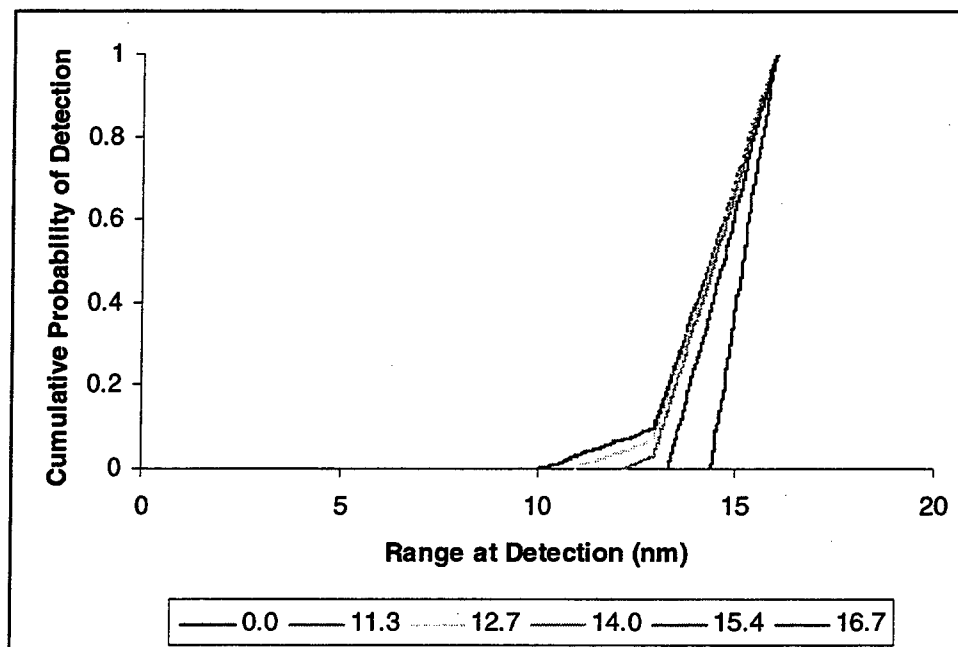
**Figure 17. Probability of Detection vs Trajectory Offset Angle for Moving Sensor, North-South Scenario**



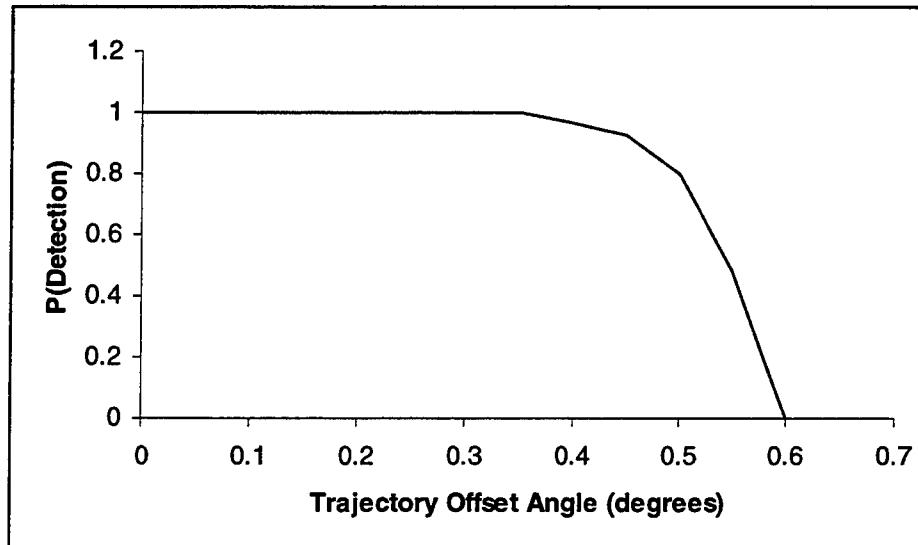
**Figure 18. Estimated Cumulative Probability of Detection vs Range for Moving Target, East-West Scenario**



**Figure 19. Probability of Detection vs Trajectory Offset Angle for Moving Sensor, East-West Scenario**



**Figure 20. Estimated Cumulative Probability of Detection vs Range for Moving Sensor, Oblique Scenario**



**Figure 21. Probability of Detection vs Trajectory Offset Angle for Moving Sensor, Oblique Scenario**

## 5.4 Verification of Sensors with Weather

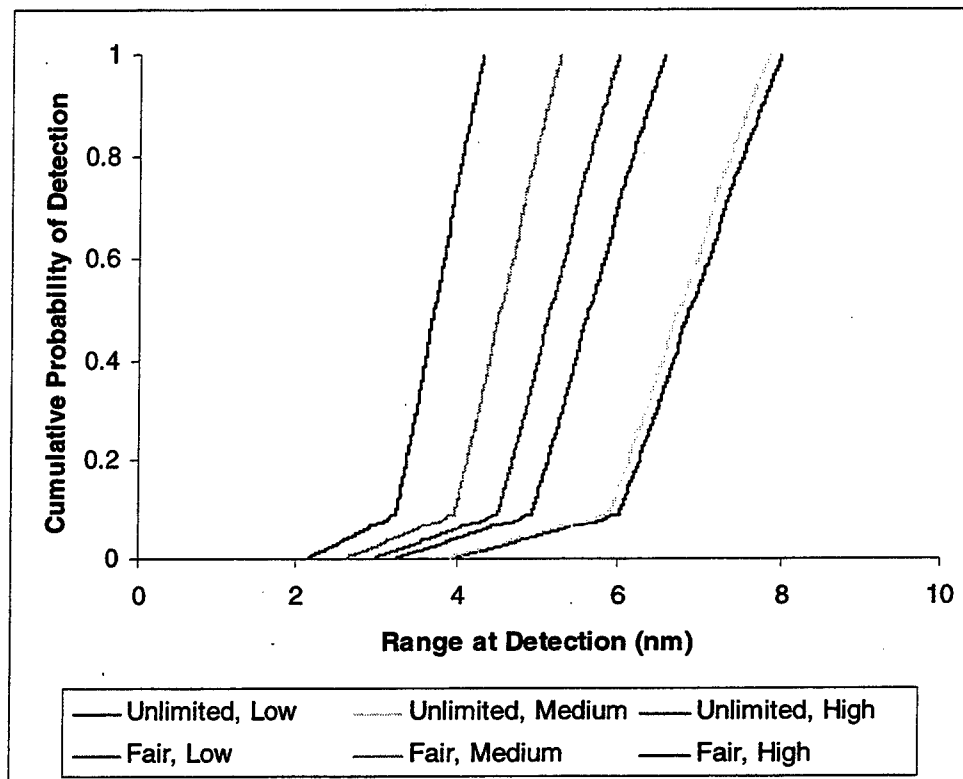
A sensor's probability/range curve (and thus its behavior and detection properties) may be modified by the weather state (Ref [4]). Sea state and precipitation can degrade the detection range of a sensor such as RADAR. Similarly, visibility can degrade a sensor such as BINOCULARS. Weather state affects a RADAR's CPOD/Range curve in qualitatively the same way, by multiplying the detection range by a quantity determined by the particular weather state and the particular sensor. On the other hand, a visual sensor has an entirely different CPOD/Range table specified based on the visibility state of the model. It is necessary to test whether each type of sensor's detection behavior is appropriately degraded by specific weather states. This was done by empirically estimating the resulting CPOD/Range curve by repeated runs.

To test the impact of sea state and visibility on RADAR sensors, runs of 1,000 iterations were made of a single target against a single sensor. The range at which detection occurred was recorded, as in the previous section. This time, however, the target was run directly at the sensor so that there was 100% detection rate. The sea state (as measured by wave height) had levels low (1.9m waves), medium (3m waves) and high (4m waves), while the visibility was either Unlimited (>25 nm) or "Fair" (2-5 nm).

Without degradation, the CPOD/Range curve for the sensor tested was piecewise linear with a minimum range of 4 nm, cumulative probability of 0.90 for greater than 6 nm, and a maximum detection range of 8 nm. Thus, the CDF for the detection range is piecewise linear with points at (4, 0.0), (6, 0.10), and (8, 1.0), respectively. MarOpsSim applies degradation to the RADAR sensor by multiplying the ranges by a factor based on both the visibility and the sea state. The factors used in these runs are shown in Table 4.

Visibility	Sea State		
	Low	Medium	High
Unlimited	1.00	0.98	0.82
Fair	0.75	0.66	0.54

**Table 4. Radar Degradation Factors for Different Sea States and Visibility**



**Figure 22. Cumulative Probability of Detection vs Range for Different Values of (Sea State, Visibility)**

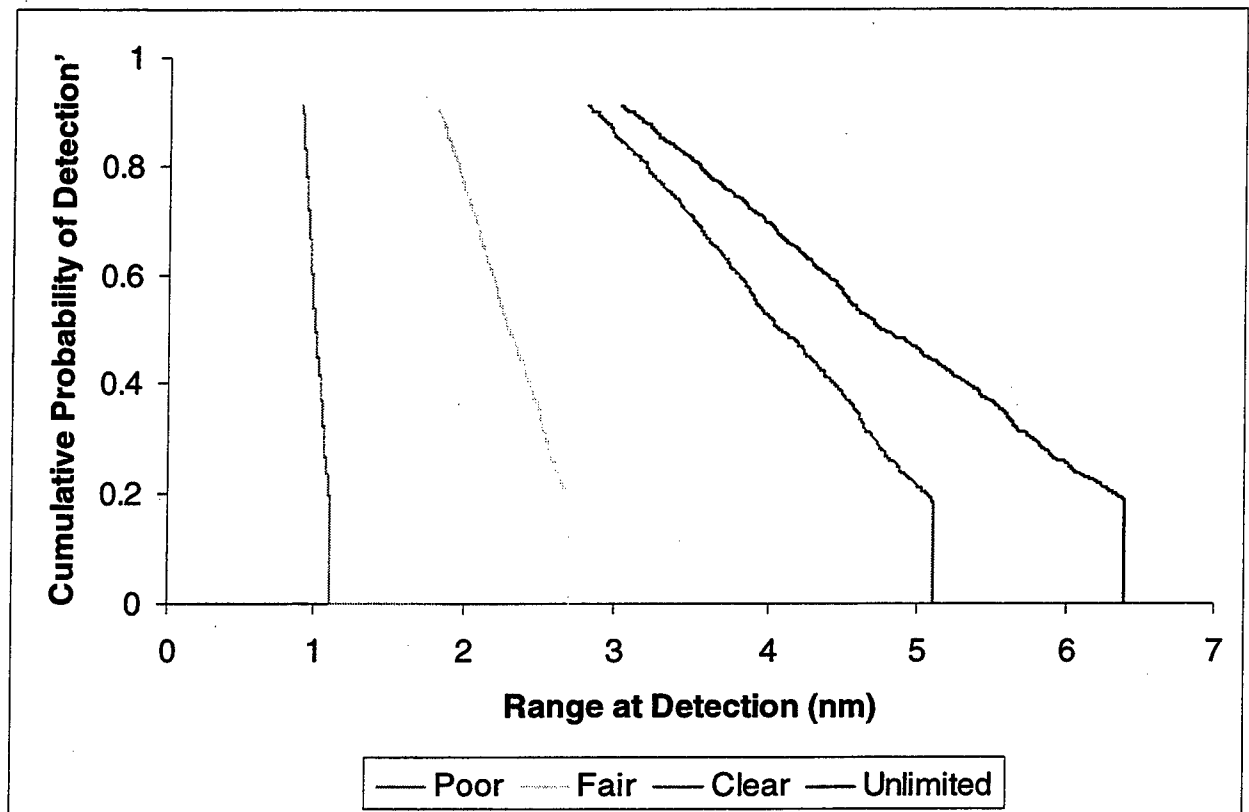
The results of the runs are shown in Figure 22. By comparing the numbers in Table 4 with the breakpoints of 4, 6, and 8 nm, it is seen that the impact of weather is working correctly. For example, for fair visibility in low sea state conditions, the breakpoints in Figure 22 are seen to be 3, 4.5, and 6 nm, respectively. These numbers are the same as multiplying the degradation factor in Table 4 of 0.75 by the original breakpoints of 4, 6, and 8 nm. The remainder of the curves are seen to be correct as well, being modified by the appropriate factor from Table 4.

Visual sensors, such as BINOCULARS, are affected only by visibility, as indicated in the CPOD'/Range table in the scripts database. Unlike RADAR, each visibility state produces an entirely new CPOD'/Range table for the visual sensor. The application of the visibility state to the CPOD'/Range for a visual sensor was tested by running 1,000 iterations of a single target against a single sensor, as before. Visibilities of 2, 5, 10, and 25 nm were used. The CPOD'/Range tables for these visibilities are shown in Table 5. The probabilities in the left column are the respective probabilities that the detection occurs at the given range or greater. For example, with a visibility of 10 nm, 50% of the time a target will be detected at a range of 4.1 nm or greater. Note that the resulting probability distribution is "defective" in that it does not have total probability mass of 1.0. There is only 0.90 probability mass, the remaining 0.10 being the probability that the target is not detected at all during an engagement. The last row of Table 5 represents a probability mass at the given values; for example, with visibility of 10 nm, approximately 20% of the interactions will result in a detection at exactly 5.1 nm.

Cumulative Probability of Detection' (CPOD')	Visibility			
	2 nm (Poor)	5 nm (Fair)	10 nm (Clear)	25 nm (Unlimited)
0.90	0.9 nm	1.8 nm	2.8 nm	3.0 nm
0.50	1.0 nm	2.3 nm	4.1 nm	4.8 nm
0.20	1.1 nm	2.7 nm	5.1 nm	6.4 nm

**Table 5. CPOD'/Range Values for Visual Sensor under Different Visibilities**





**Figure 23. Cumulative Probability of Detection' vs Range for Visual Sensor under Different Visibility Conditions**

The results for the four visibilities are shown in Figure 23. Note that the cumulative probabilities in Figure 23 only goes to approximately 0.90, consistent with the 0.10 probability of no detection. Although the slopes are nearly linear, there is a slight change at exactly the breakpoints indicated in the CPOD'/Range table values shown in Table 5. Thus, the visibility state is modifying the CPOD'/Range probabilities for visual sensors correctly. The vertical parts of the probability curves in Figure 23 correspond to the probability of 0.20 for detections at the ranges in the last row of Table 5. The heights are approximately 0.20, indicating that the algorithm is working properly.

The impact of weather has been shown to follow the described algorithms for degrading the detection of targets, whether by applying a degradation factor, in the case of RADAR, or by a new CPOD'/Range table based on the weather state value, as with the visual sensor.

## **6 Verification of Script Parsing**

As mentioned previously, MarOpsSim uses a combination of compiled code together with scripts to produce its runs. The scripts are coded in plain text files according to a flexible syntax. Scripts are used to depict things from tactics to certain characteristics of assets and targets. The scripts are parsed into an input database file. The database entries consist of sequential primitive execution statements that are performed by the executable part of the program. Since scripts play a key role in MarOpsSim scenarios, it is important that they be correctly parsed in to the database.

Verification of the script parsing was done throughout the runs mentioned above that were intended to test random number generation, platform movement, and sensing. Each of those scenarios required that at least simple scripts be correctly parsed and executed.

In every case for more than 100 distinct scenarios, the scripts were correctly parsed into the database and correctly executed. The script parser is thus verified to be working correctly in processing and executing the script files.

## 7 Conclusions

The MarOpsSim Core Validation has examined properties of MarOpsSim that are fundamental to all future development. These areas included random number generation, event list processing, unrestricted and restricted platform movement, and sensing with and without weather.

The random number generator was found to produce streams of numbers that are statistically indistinguishable from independent identically distributed random variates. The Event List processing was implemented correctly and reliably scheduled events in the correct temporal order. Platform movement was performed by a uniform, linear movement algorithm. The implementation moved the entities to the correct location at the correct time and avoided land masses when appropriate. This type of motion is appropriate for a wide range of scenarios, particularly those for which MarOpsSim is most likely to be used. If necessary, more complicated movement algorithms may easily be implemented in MarOpsSim. The sensing algorithms give much flexibility to the modeler and may be parameterized to match nearly any type of active sensor. Weather, sea state, and visibility were all capable of modifying the detection properties of a sensor. The algorithms were found to be correctly implemented, producing results which matched the statistical properties for the various parameters run.

The fundamental core algorithms were validated rather than any specific assets or entities. These basic core functions have been thoroughly tested and are operating as designed. The execution of the core functionality of MarOpsSim, a critical first step towards the ultimate V&V of the MarOpsSim model for Deepwater Acquisition applications, is correct and can be considered complete.

## 8 References

- [1] Knuth, D. (1998) *The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Third Edition*, Addison-Wesley, Reading, MA.
- [2] Law, A and D. Kelton (1992) *Simulation Modeling and Analysis, 2<sup>nd</sup> Edition*, McGraw Hill, New York, NY.
- [3] *United States Coast Guard Deepwater Capability Replacement Analysis, Modeling and Simulation Master Plan (MSMP)* (September 1998)
- [4] *USCG Maritime Operations Simulation (MarOpsSim) Consolidated Software Design Document Draft* (Working Document 1999)

## DISTRIBUTION LIST

1. Research Office (Code 09)..... 1  
Naval Postgraduate School  
Monterey, CA 93943-5000
2. Dudley Knox Library (Code 013)..... 2  
Naval Postgraduate School  
Monterey, CA 93943-5002
3. Defense Technical Information Center ..... 2  
8725 John J. Kingman Rd., STE 0944  
Ft. Belvoir, VA 22060-6218
4. Therese Bilodeau (Editorial Assistant) ..... 1  
Dept of Operations Research  
Naval Postgraduate School  
Monterey, CA 93943-5000
5. Prof. Arnold Buss (Code OR/Bu) ..... 1  
Dept of Operations Research  
Naval Postgraduate School  
Monterey, CA 93943-5000
6. Prof. Thomas Halwachs (Code OR/Ha)..... 1  
Dept of Operations Research  
Naval Postgraduate School  
Monterey, CA 93943-5000
7. Commandant (G-ADW)..... 3  
US Coast Guard  
2100 Second St. SW  
Washington, DC 20593  
Attn: CDR Lee Jacobs
8. Commandant (G-ORP) ..... 2  
US Coast Guard  
2100 Second St. SW  
Washington, DC 20593  
Attn: LCDR Christopher Rodriguez
9. MicroSystems Integration, Inc. .... 2  
158 South Broad Street, Suite C  
Pawcatuck, CT 06379-1925
10. MicroSystems Integration, Inc. .... 1  
158 South Broad Street, Suite C  
Pawcatuck, CT 06379-1925  
Attn: LCDR Phil Muir
11. US Coast Guard Research & Development Center..... 3  
Maritime Operations Technology Division  
1082 Shennecossett Rd.  
Groton, CT 06340  
Attn: Dr. Keith Gross